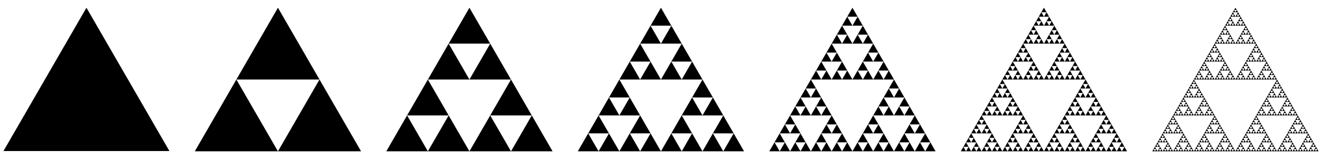
Trójkąt Sierpińskiego gra o życie

*Karol GRYSZKA**

* Uniwersytet Pedagogiczny w Krakowie, Oddział Krakowski PTM

Tytuł niniejszego artykułu jest zestawieniem dwóch pozornie odległych pojęć matematycznych. Pierwszym z nich jest *trójkąt Sierpińskiego* – jeden z najlepiej rozpoznawalnych fraktali. Drugim jest *gra w życie* – automat komórkowy opisany w 1970 roku przez Johna Conwaya.

Konstrukcja trójkąta Sierpińskiego została podana przez Wacława Sierpińskiego w 1915 roku, choć już wcześniej obserwowano mozaiki i inne wzory o podobnym kształcie. Ten fraktal powstaje w następujący sposób: dowolny trójkąt (u Sierpińskiego równoboczny) dzielony jest na cztery mniejsze identyczne trójkąty, z których środkowy jest usuwany; na pozostałych trzech trójkątach cały proces dzielenia i usuwania środkowej części powtarza się w nieskończoność. Powstała „na końcu" figura to właśnie trójkąt Sierpińskiego. Na rysunku poniżej przedstawionych jest pierwszych sześć etapów konstrukcji, gdy początkowy trójkąt jest równoboczny.



Trójkąt Sierpińskiego ma wiele zaskakujących związków z innymi obiektami matematycznymi, takimi jak trójkąt Pascala, gra w chaos, L-systemy, iterowane systemy funkcji czy popularna łamigłówka „Wieże Hanoi". My przedstawimy jego związek z grą w życie.

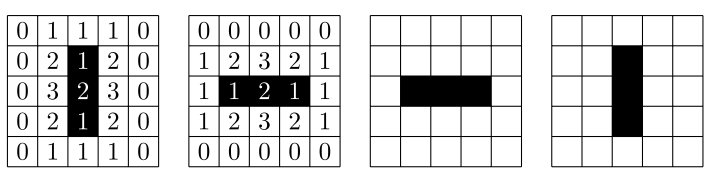
Tytułowy trójkąt można opisać również w bardziej finezyjny sposób. Niech A, B oraz C będą trzema niewspółliniowymi punktami. Trójkątem Sierpińskiego o wierzchołkach w tych punktach nazywamy zbiór S_{ABC} wszystkich punków postaci $uA+vB+wC$, spełniających warunki:
- $u, v, w \in [0,1]$,
- $u+v+w=1$,
- jeśli weźmiemy rozwinięcia **dwójkowe** liczb u, v i w, czyli $u = 0,u_1u_2\ldots_{(2)}$, $v = 0,v_1v_2\ldots_{(2)}$, $w = 0,w_1w_2\ldots_{(2)}$, to $u_i+v_i+w_i = 1$ dla wszystkich $i \geqslant 1$.

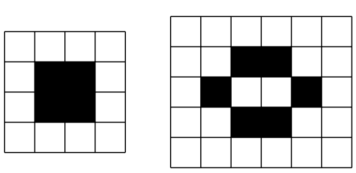
Przykłady rozwinięć dwójkowych liczb niecałkowitych: $\frac{1}{2} = 0,1_{(2)}$, $\frac{1}{3} = 0,(01)_{(2)}$, $\sqrt{2} = 1,0110101000001001\ldots_{(2)}$. Zauważmy, że środek ciężkości trójkąta ABC nie należy do S_{ABC}, z drugiej strony każdy punkt odcinka AB należy do S_{ABC} (dlaczego?).

Gra w życie jest algorytmem opisanym następującymi regułami:

- gra rozgrywa się na nieskończonej płaskiej planszy, podzielonej na identyczne kwadratowe komórki; w szczególności każda komórka **sąsiaduje** z ośmioma komórkami dokoła niej,
- każda komórka znajduje się w jednym z dwóch stanów: jest albo żywa, albo martwa,
- gra odbywa się w turach (układ ewoluuje); w każdej turze tworzony jest nowy układ (generacja) komórek na podstawie poprzedniej tury,
- gdy martwa komórka uzyska dokładnie trzech żywych sąsiadów, staje się żywa w kolejnej generacji – tak powstaje nowe życie,
- żywa komórka posiadająca dokładnie dwóch lub trzech sąsiadów przeżywa do następnej generacji; w przeciwnym razie staje się martwa.

Powyższe reguły pozwalają na tworzenie ogromnej liczby bardzo ciekawych wzorów. Na rysunku poniżej przedstawiony jest prosty przebieg gry dla układu trzech żywych komórek (oznaczone czarnym kolorem).



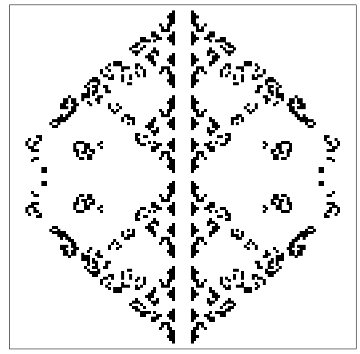


Liczby w kwadratach oznaczają liczbę żywych sąsiadów danej komórki. W tym przykładzie generacje ewoluują okresowo – czyli w którejś z tur powtarza się układ, który wystąpił już wcześniej. Generacja jest zaś stała, gdy w kolejnych ewolucjach układy żywych i martwych komórek są identyczne (rysunki na marginesie).

Automat komórkowy to zbiór komórek tworzących siatkę (komórki mogą być kwadratowe, sześciokątne, ale również sześcienne, w kształcie czworościanu i innych figur), które mogą przyjmować jeden ze z góry zadanych stanów. Układ komórek podlega ewolucji w czasie zgodnie z regułami opisanymi przez tak zwaną funkcję przejścia. Każda ewolucja tworzy nowe generacje komórek począwszy od pewnego ustalonego stanu początkowego.

Gra w życie jest szczególnym przypadkiem automatu komórkowego – przykładem ogromnej klasy algorytmów komputerowych, które zostały również sformalizowane w języku matematyki.

Bogactwo różnych schematów, układów okresowych i wielu obrazów wygenerowanych w grze w życie sprawia, że mający prawie pół wieku algorytm nadal jest popularny (istnieje choćby cała grupa ludzi, którzy tworzą coraz ciekawsze konstrukcje z użyciem współczesnych programów symulujących grę – jednym z takich programów jest *golly*, dostępny w internecie na stronie golly.sourceforge.net) i nie zaszkodziły słowa Johna Conwaya – twórca gry wielokrotnie przyznawał, że nie lubi własnego pomysłu.

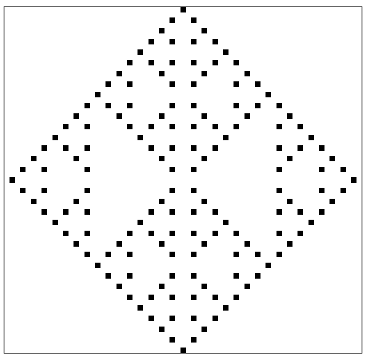


Co łączy trójkąt Sierpińskiego z grą w życie? Otóż ten fraktal można wygenerować w grze Conwaya. Ściślej – nie sam fraktal, a jedynie jego zarys, który oddaje przybliżoną strukturę jednego z etapów jego konstrukcji. Tę strukturę można otrzymać, gdy początkowym stanem będzie długa „kreska” (czyli długi rząd żywych komórek). Wtedy pewna generacja takiego układu będzie przypominać fraktal.

Spójrzmy na przykład po prawej stronie. Nie jest oczywiste, że rysunek odzwierciedla fraktal (a w istocie – dwie jego kopie zestawione podstawami) – wzór powstał z generacji złożonej ze 130 komórek ustawionych w linii poziomej i jest efektem 65 ewolucji. Duże odstępstwo od właściwego kształtu, szczególnie w okolicy prawego i lewego końca rysunku, wynika z małej skali generacji początkowej (mało żywych komórek).

Dużo lepsze rezultaty otrzymamy, gdy stanem początkowym będzie kilka-kilkadziesiąt tysięcy komórek ustawionych w jednej linii. Film `www.youtube.com/watch?v=4OSW6kfAnPI` pokazuje, że „wystarczy” około 15 000 komórek, aby otrzymać wyraźny obraz fraktala. Własne eksperymenty można prowadzić również we wspomnianym wcześniej programie *golly*.

Reguły gry Conwaya można modyfikować według uznania. Klasyczne zasady zmiany stanu komórek (opisane wcześniej) oznacza się przez B3S23. Zapis interpretujemy następująco: B3 oznacza warunek rodzenia się nowej komórki (rodzi się, gdy ma dokładnie 3 żywych sąsiadów; *is **b**orn*); S23 to warunek przeżywalności (2 lub 3 żywych sąsiadów gwarantuje przeżycie; ***s**urvives*). Zastosujmy inną regułę B3S02 do stanu początkowego 66 żywych komórek ułożonych w linii prostej. Efekt takiej gry okazuje się odzwierciedlać trójkąt Sierpińskiego w stopniu znacznie dokładniejszym niż reguła B3S23. Rysunek obok przedstawia efekt po kilkudziesięciu generacjach, który jest jednocześnie stanem stałym – każda kolejna generacja jest identyczna z tą widoczną na rysunku.



Niezwykłe jest powiązanie dwóch odległych od siebie obiektów w matematyce. Dostrzeżenie takiego powiązania jest wspaniałym doświadczeniem. Jest nim niewątpliwie połączenie fraktali z automatami komórkowymi. Trójkąt Sierpińskiego może więc grać w życie – i to dosłownie. Przetrwa bowiem jako generacja ewoluująca okresowo (w regule B2S23) albo jako generacja stała (reguła B3S02).

Czy funkcja może być brudna, czyli kilka słów o programowaniu funkcyjnym

*Marcin BORKOWSKI**

* Usługi Umysłem, z inicjatywy Oddziału Poznańskiego PTM

Każdy czytelnik *Delty* wie, że jednym z podstawowych pojęć w matematyce jest *funkcja*. Matematycy nie tylko odmieniają to słowo przez wszystkie przypadki (może z wyjątkiem wołacza), ale również tworzą od niego słowa pochodne (mamy wszak równania *funkcyjne* czy analizę *funkcjonalną*).

Część czytelników *Delty* wie również, że programiści nie pozostają matematykom dłużni – *funkcje* zrobiły w programowaniu doprawdy zawrotną karierę i są obecne w zdecydowanej większości języków programowania. Spróbujemy wyjaśnić, czym różni się „funkcja” matematyka od „funkcji” programisty.

Formalna definicja funkcji, doskonale znana studentom I roku matematyki, opiera się na teorii mnogości. Funkcja jest po prostu dość specyficznym rodzajem zbioru, a mianowicie relacją (czyli pewnym zbiorem par uporządkowanych) lewostronnie całkowitą i prawostronnie jednoznaczną. Pozwolimy sobie pominąć definicje, które czytelnikom *Delty* mogą być znane, a nam są zbędne.

Oczywiście, jak to zwykle bywa, definicje swoje, a życie swoje. Gdy podsłuchamy rozmowy matematyków, okaże się, że *funkcja* to dla nich nie specjalnie spreparowany zbiór odpowiednich par, ale prawie żywa istota, która może wykonywać różne czynności. Oto jedna funkcja *rośnie*, gdy inna *maleje*. Niektóre funkcje *osiągają* swoje kresy, inne zaś *uciekają* do nieskończoności. Mało tego, zdarza się nawet, że funkcja *znika* w jakimś punkcie! Sposobów myślenia o funkcjach jest wiele. My skupimy się na jednym z nich, który jest chyba najbliższy praktyce programistycznej (co nie znaczy, że z nią tożsamy). Otóż funkcję możemy rozumieć jako model *czynności obliczania* czegoś – mówiąc językiem szkolnym, jako „wzór”.

Z takim rozumieniem funkcji matematycy mają jednak kłopoty. Po pierwsze, nie każdą funkcję można opisać wzorem. Jest tak choćby dlatego, że „wzorów” jest przeliczalnie wiele, a rodziny funkcji o dziedzinach i przeciwdziedzinach nieskończonych są nieprzeliczalne. Gdybyśmy nawet ograniczyli rozważania tylko do funkcji dających się opisać wzorem (czego matematycy z różnych powodów nie chcą robić), nie rozwiąże to wszystkich problemów. Świetnie to widać na przykładzie *porównywania* funkcji. Zapytajmy w szczególności, czy funkcje o wzorach $f(x) = (x+1)^2$ i $g(x) = x^2 + 2x + 1$ są równe, innymi słowy, czy symbole f i g oznaczają tę samą funkcję. Matematyk powie oczywiście, że tak – ale przecież te wzory nie są identyczne! Co gorsza, nierzadko zdarza się, że jest bardzo trudno orzec, czy dwa (różnie wyglądające) wzory opisują tę samą funkcję. Formalnie, jest to problem nierozstrzygalny. Intuicję tego pojęcia ma